Decompositions of All Different, Global Cardinality and Related Constraints

Christian Bessiere[*] LIRMM, CNRS Montpellier bessiere@lirmm.fr
George Katsirelos[†] NICTA, Sydney gkatsi@gmail.com
Nina Narodytska[†] NICTA and UNSW Sydney ninan@cse.unsw.edu.au
Claude-Guy Quimper EPM, Montréal cquimper@gmail.com
Toby Walsh[†] NICTA and UNSW Sydney toby.walsh@nicta.com.au

Abstract

We show that some common and important global constraints like ALL-DIFFERENT and GCC can be decomposed into simple arithmetic constraints on which we achieve bound or range consistency, and in some cases even greater pruning. These decompositions can be easily added to new solvers. They also provide other constraints with access to the state of the propagator by sharing of variables. Such sharing can be used to improve propagation between constraints. We report experiments with our decomposition in a pseudo-Boolean solver.

1 Introduction

Global constraints allow users to specify patterns that commonly occur in problems. One of the oldest and most useful is the ALL-DIFFERENT constraint [Lauriere, 1978; Régin, 1994]. This ensures that a set of variables are pairwise different. Global constraints can often be decomposed into more primitive constraints. For example, the ALL-DIFFERENT constraint can be decomposed into a clique of binary inequalities. However, such decompositions usually do not provide a global view and are thus not able to achieve levels of local consistency, such as bound and domain consistency. Considerable effort has therefore been invested in developing efficient propagation algorithms to reason globally about such constraints. For instance, several different propagation algorithms have been developed for the ALL-DIFFERENT constraint [Régin, 1994; Leconte, 1996; Puget, 1998; Mehlhorn and Thiel, 2000; Lopez-Ortiz *et al.*, 2003]. In this paper, we show that several important global constraints including ALL-DIFFERENT can be decomposed into simple arithmetic constraints whilst still providing a global view since bound consistency can be achieved.

There are many reasons why such decompositions are interesting. First, it is very surprising that complex propagation algorithms can be simulated by simple decompositions. In many cases, we show that reasoning with the decompositions is of similar complexity to existing monolithic propagation algorithms. Second, these decompositions can be easily added to a new solver. For example, we report experiments here using these decompositions in a state of the art pseudo-Boolean solver. We could just as easily use them in an ILP solver. Third, introduced variables in these decompositions give access to the state of the propagator. Sharing of such variables between decompositions can increase propagation. Fourth, these decomposition provide a fresh perspective to propagating global constraints that may be useful. For instance, our decompositions of the ALL-DIFFERENT constraint suggest learning nogoods based on small Hall intervals.

[*]Supported by the project ANR-06-BLAN-0383-02.

[†]NICTA is funded by the Australian Government through the Department of Broadband, Communications and the Digital Economy and the Australian Research Council.

2 Formal Background

A constraint satisfaction problem (CSP) consists of a set of variables, each with a finite domain of values, and a set of constraints specifying allowed combinations of values for some subset of variables. We use capitals for variables and lower case for values. We write $dom(X)$ for the domain of possible values for X, $min(X)$ for the smallest value in $dom(X)$, $max(X)$ for the greatest, and $range(X)$ for the interval $[min(X), max(X)]$. A *global constraint* is one in which the number of variables n is a parameter. For instance, the global ALL-DIFFERENT$([X_1, \ldots, X_n])$ constraint ensures that $X_i \neq X_j$ for any $i < j$ [Régin, 1994]. We will assume values range over 1 to d.

Constraint solvers typically use backtracking search to explore the space of partial assignments. After each assignment, propagation algorithms prune the search space by enforcing local consistency properties like domain or bound consistency. A constraint is *domain consistent* (*DC*) iff when a variable is assigned any of the values in its domain, there exist compatible values in the domains of all the other variables of the constraint. Such an assignment is called a *support*. A constraint is *bound consistent* (*BC*) iff when a variable is assigned the minimum or maximum value in its domain, there exist compatible values between the minimum and maximum domain value for all the other variables. Such an assignment is called a *bound support*. Finally, between domain and bound consistency is range consistency. A constraint is *range consistent* (*RC*) iff when a variable is assigned any value in its domain, there exists a bound support.

Constraint solvers usually enforce local consistency after

each assignment down any branch in the search tree. For this reason, it is meaningful to compute the total amortised cost of enforcing a local consistency down an entire branch of the search tree so as to capture the incremental cost of propagation. We will compute complexities in this way.

3 ALL-DIFFERENT constraint

The ALL-DIFFERENT constraint is one of the most useful global constraints available to the constraint programmer. For instance, it can be used to specify that activities sharing the same resource take place at different times. A central concept in propagating the ALL-DIFFERENT constraint is the notion of a *Hall interval*. This is an interval of m domain values which completely contains the domains of m variables. $[a, b]$ is a Hall interval iff $|\{i \mid dom(X_i) \subseteq [a, b]\}| = b - a + 1$. In any bound support, the variables whose domains are contained within the Hall interval consume all the values in the Hall interval, whilst any other variables must find their support outside the Hall interval.

Example 1. *Consider an* ALL-DIFFERENT *constraint over the following variables and values:*

	1	2	3	4	5
X_1			*	*	
X_2	*	*	*	*	
X_3			*	*	
X_4		*	*	*	*
X_5	*				

$[1, 1]$ is a Hall interval of size 1 as the domain of 1 variable, X_5 is completely contained within it. Therefore we can remove $[1, 1]$ from the domains of all the other variables. This leaves X_2 with a domain containing values 2,3,4.

$[3, 4]$ is a Hall interval of size 2 as it completely contains the domains of 2 variables, X_1 and X_3. We can thus remove $[3, 4]$ from the domains of X_2 and X_4. This leaves the following range consistent domains:

	1	2	3	4	5
X_1			*	*	
X_2		*			
X_3			*	*	
X_4		*			*
X_5	*				

Enforcing bound consistency on the same problem does not create holes in domains. That is, it would leave X_4 with the values 2,3,4,5.

To identify and prune such Hall intervals from the domains of other variables, Leconte has proposed a RC propagator for the ALL-DIFFERENT constraint [Leconte, 1996] that runs in $\Theta(n^2)$ time. We now propose a simple decomposition of the ALL-DIFFERENT constraint which permits us to enforce RC. The decomposition ensures that no interval can contain more variables than its size. We introduce $O(nd^2)$ new 0/1 variables, A_{ilu} to represent whether X_i takes a value in the interval $[l, u]$. For $1 \leq i \leq n$, $1 \leq l \leq u \leq d$ and $u - l < n$, we post the following constraints:

$$A_{ilu} = 1 \iff X_i \in [l, u] \quad (1)$$

$$\sum_{i=1}^{n} A_{ilu} \leq u - l + 1 \quad (2)$$

We illustrate this decomposition on our running example.

Example 2. *Consider again the last example (i.e. an* ALL-DIFFERENT *constraint on $X_1 \in [3, 4]$, $X_2 \in [1, 4]$, $X_3 \in [3, 4]$, $X_4 \in [2, 5]$ and $X_5 \in [1, 1]$).*

First take the interval $[1, 1]$. Since $X_5 \in [1, 1]$, (1) implies $A_{511} = 1$. Now from (2), $\sum_{i=1}^{4} A_{i11} \leq 1$. That is, at most one variable can take a value within this interval. This means that $A_{211} = 0$. Using (1) and $A_{211} = 0$, we get $X_2 \notin [1, 1]$. Since $X_2 \in [1, 4]$, this leaves $X_2 \in [2, 4]$.

Now take the interval $[3, 4]$. From (1), $A_{134} = A_{334} = 1$. Now from (2), $\sum_{i=1}^{4} A_{i34} \leq 2$. That is, at most 2 variables can take a value within this interval. This means that $A_{234} = A_{434} = 0$. Using (1) we get $X_2 \notin [3, 4]$, $X_4 \notin [3, 4]$. Since $X_2 \in [2, 4]$ and $X_4 \in [2, 5]$, this leaves $X_2 = 2$ and $X_4 \in \{2, 5\}$. Local reasoning about the decomposition has thus made the original ALL-DIFFERENT *constraint range consistent.*

We will prove that enforcing DC on the decomposition enforces RC on the original ALL-DIFFERENT constraint. We find it surprising that a simple decomposition like this can simulate a complex propagation algorithm like Leconte's. In addition, the overall complexity of reasoning with the decomposition is similar to Leconte's propagator.

Theorem 1. *Enforcing DC on constraints (1) and (2) enforces RC on the corresponding* ALL-DIFFERENT *constraint in $O(nd^3)$ down any branch of the search tree.*

Proof: [Leconte, 1996] provides a necessary and sufficient condition for RC of the ALL-DIFFERENT constraint: every Hall interval should be removed from the domain of variables whose domains are not fully contained within that Hall interval. Let $[a, b]$ be a Hall interval. That is, $|H| = b - a + 1$ where $H = \{i \mid dom(X_i) \subseteq [a, b]\}$. Constraint (1) fixes $A_{iab} = 1$ for all $i \in H$. The inequality (2) with $l = a$ and $u = b$ becomes tight fixing $A_{iab} = 0$ for all $i \notin H$. Constraint (1) for $l = a$, $u = b$, and $i \notin H$ removes the interval $[a, b]$ from the domain of X_i as required for RC.

There are $O(nd^2)$ constraints (1) that can be woken $O(d)$ times down the branch of the search tree. Each propagation requires $O(1)$ time. Constraints (1) therefore take $O(nd^3)$ down the branch of the search tree to propagate. There are $O(d^2)$ constraints (2) that each take $O(n)$ time to propagate down the branch of the search tree for a total of $O(nd^2)$ time. The total running time is given by $O(nd^3) + O(nd^2) = O(nd^3)$. □

Note that if we use a solver in which we can specify that constraints only wake on reaching a particular bound, we can decrease to $O(1)$ the number of times constraints (1) are woken, which gives a total complexity in $O(nd^2)$.

What about bound consistency of the ALL-DIFFERENT constraint? By using a representation that can only prune bounds [Ohrimenko *et al.*, 2007], we can give a decomposition that achieves BC in a similar way. In addition, we can reduce the overall complexity in the case that constraints are woken whenever their bounds change. We introduce new 0/1 variables, B_{ik}, $1 \leq k \leq d$ and replace (1) by the following constraints:

$$B_{il} = 1 \iff X_i \leq l \quad (3)$$

$$A_{ilu} = 1 \iff (B_{i(l-1)} = 0 \wedge B_{iu} = 1) \quad (4)$$

Theorem 2. *Enforcing BC on constraints (2) to (4) enforces BC on the corresponding* ALL-DIFFERENT *constraint in $O(nd^2)$ down any branch of the search tree.*

Proof: We first observe that BC is equivalent to DC on constraints (2) because A_{ilu} are Boolean variables. So, the proof follows that for Theorem 1 except that fixing $A_{ilu} = 0$ prunes the bounds of $dom(X_i)$ if and only if $B_{i(l-1)} = 0$ or $B_{iu} = 1$, that is, if and only if exactly one bound of the domain of X_i intersects the interval $[l, u]$. Only the bounds that do not have a bound support are shrunk. The complexity reduces as (3) appears $O(nd)$ times and is woken $O(d)$ times, whilst (4) appears $O(nd^2)$ times and is woken just $O(1)$ time. □

A special case of ALL-DIFFERENT is PERMUTATION when we have the same number of values as variables, and the values are ordered consecutively. A decomposition of PERMUTATION just needs to replace (2) with the following equality where (as before) $1 \leq l \leq u \leq d$, and $u - l < n$:

$$\sum_{i=1}^{n} A_{ilu} \quad = \quad u - l + 1 \tag{5}$$

This can increase propagation. In some cases, DC on constraints (1) and (5) will prune values that a RC propagator for PERMUTATION would miss.

Example 3. *Consider a* PERMUTATION *constraint over the following variables and values:*

	1	2	3
X_1	*		*
X_2	*		*
X_3	*	*	*

These domains are range consistent. However, take the interval $[2, 2]$. By DC on (1), $A_{122} = A_{222} = 0$. Now, from (5), we have $\sum_{i=1}^{3} A_{i22} = 1$. Thus $A_{322} = 1$. By (1), this sets $X_3 = 2$. On this particular problem instance, DC on constraints (1) and (5) has enforced domain consistency on the original ALL-DIFFERENT *constraint.*

4 GCC constraint

A generalization of the ALL-DIFFERENT constraint is the global cardinality constraint, $\text{GCC}([X_1, \ldots, X_n], [l_1, \ldots, l_m], [u_1, \ldots, u_m])$. This ensures that the value i occurs between l_i and u_i times in X_1 to X_n. The GCC constraint is useful in resource allocation problems where values represent resources. For instance, in the car sequencing problem (prob001 at CSPLib.org), we can post a GCC constraint to ensure that the correct number of cars of each type is put on the assembly line.

We can decompose GCC in a similar way to ALL-DIFFERENT but with an additional $O(d^2)$ integer variables, N_{lu} to represent the number of variables using values in each interval $[l, u]$. Clearly, $N_{lu} \in [\sum_{i=l}^{u} l_i, \sum_{i=l}^{u} u_i]$ and $N_{1d} = n$. We then post the following constraints for $1 \leq i \leq n, 1 \leq l \leq u \leq d, 1 \leq k < u$:

$$A_{ilu} = 1 \iff X_i \in [l, u] \tag{6}$$

$$N_{lu} \quad = \quad \sum_{i=1}^{n} A_{ilu} \tag{7}$$

$$N_{1u} \quad = \quad N_{1k} + N_{(k+1)u} \tag{8}$$

Example 4. *Consider a* GCC *constraint with the following variables and upper and lower bounds on the occurrences of values:*

v	1	2	3	4	5
X_1	*				
X_2	*	*	*	*	*
X_3			*		
X_4	*	*	*	*	*
X_5	*	*	*	*	*
l_v	1	1	0	1	1
u_v	5	5	5	5	5

Enforcing RC removes 1 and 3 from X_2, X_4 and X_5 and leaves the other domains unchanged. We can derive this from our decomposition. From the lower and upper bounds on the number of occurrences of the values, we have $N_{ii} \in [1, 5]$ except for $N_{33} \in [0, 5]$ and we have $N_{12} \in [2, 10], N_{13} \in [2, 15]$ and $N_{14} \in [3, 20]$. By (6), $A_{333} = 1$. From (7), $N_{33} = \sum_{i=1}^{6} A_{i33} \in [1, 5]$. From $N_{15} = N_{14} + N_{55}$ we have $N_{14} \in [3, 4]$ (i.e., upper bound decreased) because $N_{15} = 5$ and $N_{55} \in [1, 5]$. Similarly, we derive from $N_{14} = N_{13} + N_{44}$ that $N_{13} \in [2, 3]$ and from $N_{13} = N_{12} + N_{33}$ that $N_{12} \in [2, 2]$. From the same constraint, we shrink N_{13} to $[3, 3]$ and N_{33} to $[1, 1]$. Finally, $N_{12} = N_{11} + N_{22}$ shrinks N_{11} to $[1, 1]$. By (6), $A_{111} = A_{333} = 1$, so by (7), $A_{i11} = 0, i, \in 2..5$ and $A_{i33} = 0, i, \in \{1, 2, 4, 5\}$. By (6), this removes 1 and 3 from X_2, X_4, X_5. Local reasoning about the decomposition has made the original GCC *constraint range consistent.*

We next show that enforcing DC on constraint (6) and BC on constraints (7) and (8) enforces RC on the GCC constraint.

Theorem 3. *Enforcing DC on constraint (6) and BC on constraints (7) and (8) achieves RC on the corresponding* GCC *constraint in $O(nd^3)$ time down any branch of the search tree.*

Proof: We use I_V for the number of variables X_i whose range $range(X_i)$ intersects the set V of values, and S_V for the number of variables X_i whose range is a subset of V. We first show that if RC fails on the GCC, DC on (6) and BC on (7) and (8) will fail. We derive from [Quimper *et al.*, 2005, Lemmas 1 and 2] that RC fails on a GCC if and only if there exists a set of values V such that $S_V > \sum_{v \in V} u_v$ or such that $I_V < \sum_{v \in V} l_v$. Suppose first a set V such that $S_V > \sum_{v \in V} u_v$. The fact that domains are considered as intervals implies that either $range(V)$ includes more variable domains than the sum of the upper bounds (like V), or the union of the $range(X_i)$ that are included in V lets a hole of unused values in V, which implies that there exists an interval $[l, u] \subset V$ such that $S_{[l,u]} > \sum_{v \in [l,u]} u_v$. So, in any case, there exists an interval $[l, u]$ in V with $S_{[l,u]} > \sum_{v \in [l,u]} u_v$. By (6) we have $\sum_{i=1}^{n} A_{ilu} \geq S_{[l,u]}$ whereas the greatest value in the domain of N_{lu} was set to $\sum_{v \in [l,u]} u_v$. So BC will fail on $N_{lu} = \sum_{i=1}^{n} A_{ilu}$. Suppose now that a set $V = \{v_1, \ldots, v_k\}$ is such that $I_V < \sum_{v_i \in V} l_{v_i}$. The total number of values taken by X_i variables being equal to n, the number of variables X_i with $range(X_i)$ not intersecting V is greater than $n - \sum_{v_i \in V} l_{v_i}$, that is, $S_{[1,v_1-1]} + S_{[v_1+1,v_2-1]} + \ldots + S_{[v_k+1,d]} > n - \sum_{v_i \in V} l_{v_i}$. Thanks to (7), we know that for any l, u, $N_{lu} \geq S_{[l,u]}$. So, $N_{1(v_1-1)} + N_{(v_1+1)(v_2-1)} + \ldots + N_{(v_k+1)d} > n - \sum_{v_i \in V} l_{v_i}$.

The initial domains of N_{lu} variables also tell us that for every v_i in V, $N_{v_iv_i} \geq l_{v_i}$. Thus, $min(N_{1(v_1-1)}) + min(N_{v_1v_1}) + min(N_{(v_1+1)(v_2-1)}) + \ldots + min(N_{(v_k+1)d}) > n = max(N_{1d})$. Successively applying BC on $N_{1v_1} = N_{1(v_1-1)} + N_{v_1v_1}$, then on $N_{1(v_2-1)} = N_{1v_1} + N_{(v_1+1)(v_2-1)}$, and so on until $N_{1d} = N_{1v_k} + N_{(v_k+1)d}$ will successively increase the minimum of these variables and will lead to a failure on N_{1d}.

We now show that when DC on (6) and BC on (7) and (8) do not fail, it prunes all values that are pruned when enforcing RC on the GCC constraint. Consider a value $v \in dom(X_q)$ for some $q \in 1..n$ such that v does not have any bound support. We derive from [Quimper *et al.*, 2005, Lemmas 1 and 6] that a value v for a variable X_q does not have a bound support on GCC if and only if there exists a set V of values such that either (i) $S_V = \sum_{w \in V} u_w$, $v \in V$ and $range(X_q)$ is not included in V, or (ii) $I_V = \sum_{w \in V} l_w$, $v \notin V$ and $range(X_q)$ intersects V. In case (i), V contains v and the values it contains will be taken by too many variables if X_q is in it. In case (ii), V does not contain v and its values will be taken by not enough variables if X_q is not in it. Consider case (i): Since DC did not fail on (6), by a similar reasoning as above for detecting failure, we derive that V is composed of intervals $[l, u]$ such that $S_{[l,u]} = \sum_{w \in [l,u]} u_w$. Consider the interval $[l, u]$ containing v. The greatest value in the initial domain of N_{lu} was $\sum_{w \in [l,u]} u_w$, which is exactly the number of variables with range included in $[l, u]$ without counting X_q because its range is not included in V. Thus, (7) forces $A_{qlu} = 0$ and (6) prunes value v from $dom(X_q)$ because $v \in [l, u]$ by assumption. Consider now case (ii): $V = \{v_1, \ldots, v_k\}$ is such that $I_V = \sum_{v_i \in V} l_{v_i}$. The total number of values taken by the X_i variables being equal to n, the number of variables X_i with $range(X_i)$ not intersecting V is equal to $n - \sum_{v_i \in V} l_{v_i}$, that is $S_{[1,v_1-1]} + S_{[v_1+1,v_2-1]} + \ldots + S_{[v_{k+1},d]} = n - \sum_{v_i \in V} l_{v_i}$. Thanks to (7), we know that for any l, u, $N_{lu} \geq S_{[l,u]}$. So, $N_{1(v_1-1)} + N_{(v_1+1)(v_2-1)} + \ldots + N_{(v_k+1)d} \geq n - \sum_{v_i \in V} l_{v_i}$. The initial domains of N_{lu} variables also tell us that for every v_i in V, $N_{v_iv_i} \geq l_{v_i}$. Thus, $min(N_{1(v_1-1)}) + min(N_{v_1v_1}) + min(N_{(v_1+1)(v_2-1)}) + \ldots + min(N_{(v_k+1)d}) \geq n = max(N_{1d})$. Successively applying BC on $N_{1v_1} = N_{1(v_1-1)} + N_{v_1v_1}$, then on $N_{1(v_2-1)} = N_{1v_1} + N_{(v_1+1)(v_2-1)}$, and so on until $N_{1v_k} = N_{1(v_k-1)} + N_{v_kv_k}$ will increase all $min(N_{1(v_i-1)})$ and $min(N_{1v_i})$, to the sum of the minimum values of the variables in the right side of each constraint so that $min(N_{1v_k}) = min(N_{1(v_1-1)}) + min(N_{v_1v_1}) + min(N_{(v_1+1)(v_2-1)}) + \ldots + min(N_{v_kv_k})$. Then, because $max(N_{1d}) = n$, BC on $N_{1d} = N_{1v_k} + N_{(v_k+1)d}$ will decrease the maximum value of N_{1v_k} and $N_{(v_k+1)d}$ to their minimum value, BC on $N_{1v_k} = N_{1(v_k-1)} + N_{v_kv_k}$ will decrease the maximum value of $N_{1(v_k-1)}$ and $N_{v_kv_k}$ to their minimum value, and so on until all $N_{(v_i+1)(v_{i+1}-1)}$ are forced to the singleton $min(N_{(v_i+1)(v_{i+1}-1)}) = S_{[v_i+1,v_{i+1}-1]}$. At this point, (7) forces $A_{j(v_i+1)(v_{i+1}-1)} = 0$ for every variable X_j with range not included in the interval $[v_i + 1, v_{i+1} - 1]$ because that interval is saturated by variables X_p in $S_{[v_i+1,v_{i+1}-1]}$, for which $A_{p(v_i+1)(v_{i+1}-1)} = 1$. By assumption value v is not in V, so there exists such an interval $[v_i + 1, v_{i+1} - 1]$ that contains v. Furthermore, $range(X_q)$ intersects V, so it is not included in $[v_i + 1, v_{i+1} - 1]$. Therefore, $A_{q(v_i+1)(v_{i+1}-1)}$ is forced to 0 and (6) prunes v from $dom(X_q)$.

There are $O(nd^2)$ constraints (6) that can be woken $O(d)$ times down the branch of the search tree in $O(1)$, so a total of $O(nd^3)$ down the branch. There are $O(d^2)$ constraints (7) which can be woken $O(n)$ times each down the branch for a total cost in $O(n)$ time down the branch. Thus a total of $O(nd^2)$. There are $O(d^2)$ constraints (8) that can be woken $O(n)$ times down the branch. Each propagation takes $O(1)$ time to execute for a total of $O(nd^2)$ time down the branch. The final complexity down the branch of the search tree is therefore $O(nd^3) + O(nd^2) + O(nd^2) = O(nd^3)$. □

What about bound consistency of the GCC constraint? As in the case of ALL-DIFFERENT, by replacing constraints (6) by constraints (3) and (4), the decomposition achieves BC.

Theorem 4. *Enforcing BC on constraints (3), (4), (7) and (8) achieves BC on the corresponding* GCC *constraint in $O(nd^2)$ time down any branch of the search tree.*

Proof: The proof follows that for Theorem 3 except that fixing $A_{ilu} = 0$ prunes the bounds of $dom(X_i)$ if and only if exactly one bound of the domain of X_i intersects the interval $[l, u]$. The complexity reduces to $O(nd^2)$ as BC on (3) and (4) is in $O(nd^2)$ (see Theorem 2) and BC on (7) and (8) is in $O(nd^2)$ (see Theorem 3). □

The best known algorithm for BC on GCC runs in $O(n)$ time at each call [Quimper *et al.*, 2005] and can be awaken $O(nd)$ times down a branch. This gives a total of $O(n^2d)$, which is greater than the $O(nd^2)$ here when $n > d$. Our decomposition is also interesting because, as we show in the next section, we can use it to combine together propagators.

5 Other global constraints

Many other global constraints that count variables or values can be decomposed in a similar way. For example, the global constraint SAME$([X_1, \ldots, X_n], [Y_1, \ldots, Y_n])$ is satisfied if and only if the Y_i variables are a permutation of the X_i variables. A monolithic flow-based propagator for this constraint is given in [Beldiceanu *et al.*, 2004]. The following decomposition encodes the SAME constraint where $1 \leq i \leq n$, $1 \leq l \leq u \leq d$, $l \leq k$ and $k < u$:

$$A_{ilu} = 1 \iff X_i \in [l, u], \quad B_{ilu} = 1 \iff Y_i \in [l, u]$$

$$N_{lu} = \textstyle\sum_{i=1}^{n} A_{ilu}, \qquad N_{lu} = \sum_{i=1}^{n} B_{ilu}$$

$$N_{1u} = N_{1k} + N_{(k+1)u}$$

This decomposition can be obtained by posting decompositions for EGCC$([X_1, \ldots, X_n], [O_1, \ldots, O_m])$ and EGCC$([Y_1, \ldots, Y_n], [O_1, \ldots, O_m])$ and eliminating common sub-expressions (EGCC is an extended form of the GCC constraint in which upper and lower bounds on occurrences of values are replaced by integer variables). This is another argument in favor of decompositions since it allows constraints to share "internal" state through common intermediate variables. Such sharing can increase propagation.

Example 5. *Consider the following example:*

	1	2	3	4	5
X_1	*	*			
X_2			*	*	*
Y_1	*	*	*		
Y_2				*	*

If we have $O_i \in [0,1]$ *for* $1 \leq i \leq 5$ *then both* $\text{EGCC}([X_1, X_2], [O_1, O_2, O_3, O_4, O_5])$ *and* $\text{EGCC}([Y_1, Y_2], [O_1, O_2, O_3, O_4, O_5])$ *are BC. However, enforcing BC on the decomposition of* $\text{SAME}([X_1, X_2], [Y_1, Y_2])$ *removes* 3 *from the domain of* X_2 *and* Y_1.

In fact, we conjecture that enforcing BC on this decomposition achieves BC on the SAME constraint itself. Similar decompositions can be given for other global constraints like NVALUE and COMMON.

6 Experimental Results

To test these decompositions, we ran experiments on pseudo-Boolean encodings (PB) of CSPs containing ALL-DIFFERENT and PERMUTATION constraints. We used the MiniSat+ 1.13 solver on an Intel Xeon 4 CPU, 2.0 Ghz, 4G RAM with a timeout of 600 seconds for each experiment. Our decompositions contain two types of constraints: SUM constraints like (2) and MEMBER constraints like (1). The SUM constraints is posted directly to the MiniSat+ solver. To encode MEMBER constraints, we use literals B_{ij} for the truth of $X_i \leq j$ [Ohrimenko *et al.*, 2007], and clauses of the form $(A_{ilu} = 1) \Leftrightarrow (B_{i(l-1)} = 0 \wedge B_{iu} = 1)$. This achieves bound consistency (Theorem 2). To increase propagation, we use a direct encoding with literals Z_{ij} for the truth of $X_i = j$ and clauses $(A_{ilu} = 0) \Rightarrow (Z_{ij} = 0)$, $j \in [l, u]$. The overall consistency achieved is therefore between BC and RC. We denote this encoding HI. To explore the impact of small Hall intervals, we also tried HI_k, a PB encoding with only those constraints (2) for which $u - l + 1 \leq k$. This detects Hall intervals of size at most k. Finally, we decomposed ALL-DIFFERENT into a clique of binary inequalities, and used a direct encoding to convert this into SAT (denoted BI).

Pigeon Hole Problems. Table 1 gives results on pigeon hole problems (PHP) with n pigeons and $n-1$ holes. Our decomposition is both faster and gives a smaller search tree compared to the BI decomposition. On such problems, detecting large Hall intervals is essential.

Double-Wheel Graceful Graphs. The second set of experiments uses double-wheel graceful graphs [Petrie and Smith, 2003]. We converted the CSP model in [Petrie and Smith, 2003] into a PB formula. This model has an ALL-DIFFERENT constraint on node labels and a PERMUTATION constraint on edge labels. For the PERMUTATION constraint we use (5). We strengthen the BI decomposition with clauses to ensure that every value appears at least once. Table 2 show that our decomposition outperforms the augmented BI decomposition on many instances. Whilst detecting large Hall intervals can greatly reduce search, in some cases the branching heuristics appear to be fooled by the extra variables introduced in the encodings.

Overall these experiments suggest that detecting Hall intervals reduces search significantly, and focusing on small Hall intervals may be best except on problems where large Hall intervals occur frequently.

n	BI $bt/\,t$	HI_1 $bt/\,t$	HI_3 $bt/\,t$	HI_5 $bt/\,t$	HI_7 $bt/\,t$	HI_9 $bt/\,t$
5	30/ **0.0**	28/ **0.0**	**4**/ **0.0**			
7	622/ **0.0**	539/ **0.0**	47/ **0.0**	**6**/ **0.0**		
9	16735/ 0.3	18455/ 0.7	522/ **0.0**	122/ **0.0**	**8**/ **0.0**	
11	998927/ 29.3	665586/ 44.8	5681/ 0.3	171/ **0.0**	180/ **0.0**	**10**/ 0.1
13	-/ -	-/ -	13876/ 0.9	2568/ 0.2	247/ **0.1**	**195**/ **0.1**
15	-/ -	-/ -	1744765/ 188.6	24109/ 2.6	1054/ 0.2	**165**/ **0.1**
17	-/ -	-/ -	-/ -	293762/ 48.0	8989/ 1.1	**4219**/ **0.6**
19	-/ -	-/ -	-/ -	107780/ 21.8	857175/ 368.0	**39713**/ **9.9**
21	-/ -	-/ -	-/ -	-/ -	550312/ 426.2	**57817**/ **33.5**

Table 1: PHP problems. t is time and bt is the number of backtracks to solve the problem.

DW_n	BI $bt/\,t$	HI_1 $bt/\,t$	HI_3 $bt/\,t$	HI_5 $bt/\,t$	HI_7 $bt/\,t$	HI_9 $bt/\,t$
3	176/ 0.1	90/ 0.1	**63**/ **0.1**			
4	30/ **0.1**	**14**/ 0.1	212/ 0.2			
5	**22**/ **0.2**	526/ 0.4	87/ 0.3	1290/ 1.7		
6	1341/ 1.0	873/ 0.9	**318**/ **0.7**	1212/ 2.9		
7	2948/ 3.6	2047/ 4.2	1710/ 3.6	1574/ 4.0	**27**/ **0.9**	
8	2418/ 5.5	724/ **2.2**	643/ 2.8	**368**/ 2.4	3955/ 19.5	
9	3378/ 8.6	1666/ 5.7	1616/ 9.0	**30**/ **1.8**	10123/ 129.7	405/ 6.5
10	19372/ 118.3	9355/ 66.2	14120/ 85.9	**10**/ **2.1**	4051/ 35.0	5709/ 71.2
11	839/ 5.4	12356/ 84.2	1556/ 13.9	**14**/ **2.4**	7456/ 105.2	5552/ 92.7

Table 2: Double-wheel graceful graphs. t is time and bt is the number of backtracks to solve the problem

7 Other Related Work

The ALL-DIFFERENT constraint first appeared in the ALICE constraint programming language [Lauriere, 1978]. Regin proposed a DC propagator that runs in $O(n^{2.5})$ time [Régin, 1994]. Leconte gave a RC propagator based on Hall intervals that runs in $O(n^2)$ time [Leconte, 1996]. Puget then developed a BC propagator also based on Hall intervals that runs in $O(n \log(n))$ time [Puget, 1998]. This was later improved by Melhorn and Thiel [Mehlhorn and Thiel, 2000] and then Lopez-Ortiz *et al.* [Lopez-Ortiz *et al.*, 2003].

The global cardinality constraint, GCC was introduced in the CHARME language [Oplobedu *et al.*, 1989]. Regin proposed a DC propagator based on network flow that runs in $O(n^2)$ time [Régin, 1996]. Katriel and Thiel proposed a BC propagator for the EGCC constraint [Katriel and Thiel, 2003]. Quimper *et al.* proved that enforcing DC on the EGCC constraint is NP-hard [Quimper *et al.*, 2004]. They also improved the time complexity to enforce DC and gave the first propagator for enforcing RC on GCC.

Many decompositions have been given for a wide range of global constraint. However, decomposition in general tends to hinder propagation. For instance, [Stergiou and Walsh, 1999] shows that the decomposition of ALL-DIFFERENT constraints into binary inequalities hinders

propagation. On the other hand, there are global constraints where decompositions have been given that do not hinder propagation. For example, Beldiceanu *et al.* identify conditions under which global constraints specified as automata can be decomposed into signature and transition constraints without hindering propagation [Beldiceanu *et al.*, 2005]. As a second example, many global constraints can be decomposed using ROOTS and RANGE which can themselves often be propagated effectively using simple decompositions [Bessiere *et al.*, 2005; Bessiere *et al.*, 2006a; Bessiere *et al.*, 2006b]. As a third example, decompositions of the REGULAR and CFG constraints have been given that do not hinder propagation [Quimper and Walsh, 2006; Quimper and Walsh, 2007; Quimper and Walsh, 2008; Bessiere *et al.*, 2008; Katsirelos *et al.*, 2008]. As a fourth example, decompositions of the SEQUENCE constraint have been shown to be effective [Brand *et al.*, 2007]. Finally, the PRECEDENCE constraint can be decomposed into ternary constraints without hindering propagation [Walsh, 2006].

8 Conclusions

We have shown that some common global constraints like ALL-DIFFERENT and GCC can be decomposed into simple arithmetic constraints whilst still maintaining a global view that achieves range or bound consistency. These decompositions are interesting for a number of reasons. First, we can easily incorporate them into other solvers. Second, the decompositions provide other constraints with access to the state of the propagator. Third, these decompositions provide a fresh perspective on propagation of global constraints. For instance, our results suggest that it may pay to focus propagation and nogood learning on small Hall intervals. Finally, these decompositions raise an important question. Are there propagation algorithms that cannot be efficiently simulated using decompositions? In [Bessiere *et al.*, 2009], we use circuit complexity to argue that a domain consistency propagator for the ALL-DIFFERENT constraint cannot be simulated using a polynomial sized decomposition.

References

[Beldiceanu *et al.*, 2004] N. Beldiceanu, I. Katriel, and S. Thiel. Filtering algorithms for the same constraint. In *1st Int. Conf. on Integration of AI and OR Techniques in CP*, 65–79, 2004.

[Beldiceanu *et al.*, 2005] N. Beldiceanu, I. Katriel, and S. Thiel. Reformulation of Global Constraints Based on Constraints Checkers Filtering algorithms for the same constraint. *Constraints*, 10(4): 339–362, 2005.

[Bessiere *et al.*, 2005] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan and T. Walsh. The Range and Roots Constraints: Specifying Counting and Occurrence Problems. In *19th Int. Joint Conf. on AI*, 60–65. 2005.

[Bessiere *et al.*, 2006a] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan and T. Walsh. The RANGE Constraint: Algorithms and Implementation. In *3rd Int. Conf. on Integration of AI and OR Techniques in CP (CP-AI-OR)*, 59–73, 2006.

[Bessiere *et al.*, 2006b] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan and T. Walsh. The ROOTS Constraint. In *12th Int. Conf. on Principles and Practices of CP (CP2006)*, 75–90. 2006.

[Bessiere *et al.*, 2008] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan and T. Walsh. SLIDE: A Useful Special Case of the CARD-PATH Constraint. In *18th European Conf. on AI*, 475–479. 2008.

[Bessiere *et al.*, 2009] C. Bessiere, G. Katsirelos, N. Narodytska and T. Walsh. Circuit Complexity and Decompositions of Global Constraints. In *21st Int. Joint Conf. on AI*. 2009.

[Brand *et al.*, 2007] S. Brand, N. Narodytska, C-G. Quimper, P. Stuckey and T. Walsh. Encodings of the SEQUENCE Constraint. In *13th Int. Conf. on Principles and Practice of CP (CP2007)*, 210–224. 2007.

[Katriel and Thiel, 2003] I. Katriel and S. Thiel. A fast bound consistency for the global cardinality constraint. In *9th Int. Conf. on Principles and Practice of CP (CP2003)*. 2003.

[Katsirelos *et al.*, 2008] G. Katsirelos, N. Narodytska and T. Walsh. The Weighted CFG Constraint. In *5th Int. Conf. on Integration of AI and OR Techniques in CP (CP-AI-OR)*, 323–327, 2008.

[Lauriere, 1978] J.L. Lauriere. Alice: A language and a program for solving combinatorial problems. *Artificial Intelligence*, 10:29–127, 1978.

[Leconte, 1996] M. Leconte. A bounds-based reduction scheme for constraints of difference. In *2nd Int. Workshop on Constraint-based Reasoning*, 1996.

[Lopez-Ortiz *et al.*, 2003] A. Lopez-Ortiz, C.G. Quimper, J. Tromp, and P. van Beek. A fast and simple algorithm for bounds consistency of the alldifferent constraint. In *18th National Conf. on AI*. AAAI 2003.

[Mehlhorn and Thiel, 2000] K. Mehlhorn and S. Thiel. Faster algorithms for bound-consistency of the sortedness and the alldifferent constraint. In *6th Int. Conf. on Principles and Practice of CP (CP2000)*, 306–319. 2000.

[Ohrimenko *et al.*, 2007] O. Ohrimenko, P.J. Stuckey, and M. Codish. Propagation = lazy clause generation. In *13th Int. Conf. Principles and Practice of CP (CP2007)*, 544–558, 2007.

[Oplobedu *et al.*, 1989] A. Oplobedu, J. Marcovitch, and Y. Tourbier. CHARME: Un langage industriel de programmation par contraintes, illustre par une application chez Renault. In *9th Int. Workshop on Expert Systems and their Applications*, 1989.

[Petrie and Smith, 2003] K.E. Petrie and B.M. Smith. Symmetry breaking in graceful graphs. In *9th Int. Conf. of Principles and Practice of CP (CP2003)*, 930–934, 2003.

[Puget, 1998] J.F. Puget. A fast algorithm for the bound consistency of alldiff constraints. In *15th National Conf. on AI*, 359–366. AAAI, 1998.

[Quimper and Walsh, 2006] C.-G. Quimper and T. Walsh. Global Grammar Constraints. In *12th Int. Conf. on Principles and Practices of CP (CP2006)*, 751–755. 2006.

[Quimper and Walsh, 2007] C.-G. Quimper and T. Walsh. Decomposing Global Grammar Constraints. In *13th Int. Conf. on Principles and Practices of CP (CP2007)*, 590–604. 2007.

[Quimper and Walsh, 2008] C.-G. Quimper and T. Walsh. Decompositions of Grammar Constraints. In *23rd National Conf. on AI*, 1567–1570. AAAI, 2008.

[Quimper *et al.*, 2004] C-.G. Quimper, P. van Beek, A. Lopez-Ortiz, and A. Golynski. Improved algorithms for the global cardinality constraint. In *10th Int. Conf. on Principles and Practice of CP (CP2004)*. 2004.

[Quimper *et al.*, 2005] C-G. Quimper, A. Golynski, A. López-Ortiz, and P. van Beek. An efficient bounds consistency algorithm for the global cardinality constraint. *Constraints*, 10(2):115–135, 2005.

[Régin, 1994] J-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *12th National Conf. on AI*, 362–367. AAAI, 1994.

[Régin, 1996] J-C. Régin. Generalized arc consistency for global cardinality constraints. In *13th National Conf. on AI*, 209–215. AAAI, 1996.

[Stergiou and Walsh, 1999] K. Stergiou and T. Walsh. The Difference All-difference Makes. In *16th Int. Joint Conf. on AI*, 414–419. 1999.

[Walsh, 2006] T. Walsh. Symmetry Breaking using Value Precedence. In *17th European Conf. on AI*, 168–172. 2006.